**Predicting Population Growth Rate Trends from Human Geography Factors**

Rohit Ramarathinam

CS 6375.002

**Introduction**

Geography has long been an interest of mine. In fact, when I was a kid, I wanted to be a geographer! Over time, my interests have expanded into languages, physical and political geography, and finally, human geography. Human geography, more specifically, is the study of how human activity affects or is affected by the earth and its surface. It includes the study of all things human: ethnicities, race, language, and population. Over the last several years, geographers have studied the relationship between such factors and the growth of population over time. They correctly deduced that the innovations and medicines that came from the Green Revolution in the 1970s was one of the major reasons life expectancy and population growth (globally) is trending up today.   
  
Such topics have always fascinated me, so I decided that combining this interest with machine learning would be a great way for me to understand how machine learning models work while combining it with something I love learning new things about. I decided predicting population growth from various human geography factors would be the best way to do so.

**Goals**

The main goal of the project was to predict population growth rate from geographical factors that it has historically been correlated with and **find the best model(s) that does so**. However, there were several other goals as well, including:

* Develop an understanding of how machine learning can be used in the scope of geography
* How different machine learning models operate and what do they optimize
* The impact and importance of each selected human geographical factor (features) on population growth rate (target)
* How tuning the hyperparameters of machine learning models boosts performance and predictions

In order to find the best models for predicting population growth rate, 9 models were implemented using scikit-learn, a Python library tailormade for machine learning. The following models were used and compared:

1. *Linear Regression*
2. *Support Vector Regression (SVR)*
3. *Ridge Regression*
4. *Lasso Regression*
5. *Decision Tree Regressor*
6. *Random Forest Regressor*
7. *Gradient Boosting*
8. *K-Nearest Neighbors*
9. *XGBoost*

**Method**

In order to analyze which models worked the best, a simple method was followed and applied to each model implementation. The steps followed are explained in more detail below:

1. ***Shorten dataset***The first thing that was done was to shorten the dataset – the CIA World Factbook dataset. There are simply too many facts in the factbook! Almost half of the factors that were present in the original dataset do not fit in the scope of the problem; “Total World Heritage Sites” is not a human geography factor and other features such as “Access to Sanitation – Improved” are strongly correlated to the overall development of a country, so features such as “Birth Rate” and “Total Fertility Rate” are much better predictors of country development [1].
2. ***Run model***The next step was to jump into the machine learning aspect of the project. Each of the 9 models would be executed on default settings. No hyperparameters would be manually set and default values would be used for them instead. This was done to see what the results would look like without any hyperparameter tuning, provide a baseline, and help decide later whether the model was complex enough to warrant hyperparameter tuning.
3. ***Scale***The next step was to implement feature selection. This is a vital step in models that did not produce great results with default settings, and would also scale the data so that one predictor variable’s large scale did not overweigh another more useful predictor variable’s smaller scale. This step was not used for all the models, but instead reserved for models that were prone to noise and weren’t performing well with default settings.
4. ***Select K best***This step was also not used on all models (notably, the tree-based models did not require this step). Since one of the goals was to find the best features that predict population growth rate, it was necessary to run the model after selecting “k” best features and seeing which “k” gave the best results.
5. ***Hyperparameter Tuning***Hyperparameter tuning was applied to all the models to see if tuning increased the results. GridSearchCV from scikit-learn was used to do this step; a grid variable was created containing the various hyperparameters each model had, and the grid search algorithm was executed to find the best hyperparameters for the model.
6. ***Compare evaluation metrics***Finally, evaluation metrics such as mean squared error (mse) and r2 score were implemented as a primary way of comparing the performance of the models. The goal was to get a minimum possible mse and a maximum possible r2 (without overfitting). Some room for error was desired as there are outliers in the dataset (for example, Syria has an unbelievably high birth rate, Vatican City is missing a lot of data).

**Results and Analysis**

After following the method described above, each model was then evaluated using the chosen evaluation metrics. The best obtained results are shown below.

1. ***Linear Regression****Best Obtained Model:* ***After feature selection (7 features)*** *Mean Squared Error: 0.45038780035816733  
   Mean Absolute Error: 0.3071473638851082  
   R2 Score: 0.6767349917657657*This model overfit on default settings and after feature scaling, but performed better after feature selection. The model achieved a mse of After a bit of experimenting, 7 was the best performing “k” in SelectKBest, so using the 7 best features yielded the best linear regression model. These 7 features were indices [0, 1, 2, 3, 5, 7, 14], which correspond to birth rate, total fertility rate, death rate, life expectancy at birth, infant mortality rate, gross reproduction rate, and median age.
2. ***Support Vector Regression (SVR)****Best Obtained Model:* ***After feature scaling*** *Mean Squared Error: 0.43119906624577475  
   Mean Absolute Error: 0.22149365838541224  
   R2 Score: 0.6905076700796857*This model overfit on default settings but after feature scaling, gave the best performing model (feature selection and hyperparameter tuning made minimal changes). This makes sense as support vector based models are not as sensitive to outliers and noisy features like linear regression, so all 15 features were used to predict population growth. Despite this, SVR performed only slightly better than Linear Regression, implying that the 8 features not used in the best Linear Regression had a much lesser significant impact on predicting population growth.
3. ***Ridge Regression****Best Obtained Model:* ***After feature selection (7 features)*** *Mean Squared Error: 0.45144883143161885  
   Mean Absolute Error: 0.30837781086815025  
   R2 Score: 0.6759734386810167*This model overfit on default settings and after feature scaling, but performed better after feature selection, just like linear regression. After a bit of experimenting, 7 again was the best performing “k” in SelectKBest, so using the 7 best features yielded the best ridge regression model. These 7 features were indices [0, 1, 2, 3, 5, 7, 14], which correspond to birth rate, total fertility rate, death rate, life expectancy at birth, infant mortality rate, gross reproduction rate, and median age.
4. ***Lasso Regression****Best Obtained Model:* ***After feature selection (7 features)****Mean Squared Error: 0.4904689625104833  
   Mean Absolute Error: 0.35283330112642314  
   R2 Score: 0.6479668119818055*This model also overfit on default settings and after feature scaling, but performed better after feature selection, just like linear and ridge regression. After a bit of experimenting, 7 again was the best performing “k” in SelectKBest, so using the 7 best features yielded the best ridge regression model. These 7 features were indices [0, 1, 2, 3, 5, 7, 14], which correspond to birth rate, total fertility rate, death rate, life expectancy at birth, infant mortality rate, gross reproduction rate, and median age.
5. ***Decision Tree Regressor****Best Obtained Model:* ***After hyperparameter tuning -*** *{'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 10}  
   Mean Squared Error: 0.3962480508140676  
   Mean Absolute Error: 0.3546376573617953  
   R2 Score: 0.7155936965713907*Prior to starting the project, I had a feeling the decision tree would perform well; it greedily selects the feature with the best mutual information with the target, so naturally the factors that influence population growth will be selected and be higher up in the decision tree. This was indeed true; after hyperparameter tuning, the optimal configuration {'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 10} by obtaining 0.396 for mse and an r2 score of 0.716, the best so far. Upon visualization, the top features included total fertility rate, death rate, birth rate, median age, and net migration rate.
6. ***Random Forest Regressor****Best Obtained Model:* ***Default***  *Mean Squared Error: 0.3338762255172413  
   Mean Absolute Error: 0.24958620689655173  
   R2 Score: 0.7603609584779711*The goal now was to see if bagging the decision tree model would yield better results…and it did! The default hyperparameters performed the best and gave an improved mse of 0.334 and an r2 score of 0.76. The best tree from the random forest also yielded a similar tree to the one from the decision tree. Since it generates many trees, it wasn’t an exact match; however, the top features were consistently among total fertility rate, death rate, birth rate, median age, gross reproduction rate, currently married women and net migration rate.
7. ***Gradient Boosting Regressor****Best Obtained Model:* ***After hyperparameter tuning -*** *{'n\_estimators': 150}  
   Mean Squared Error: 0.2286411378542144  
   Mean Absolute Error: 0.18710406261159715  
   R2 Score: 0.8358932474362102*Boosting is a powerful algorithm that could lower both the variance and bias of the decision tree models, so this was naturally a great model to implement. This model worked even better, yielding an mse of 0.229 and r2 of 0.836, even better than the random forest. It also returned a smaller set of factors including total fertility rate, death rate, birth rate, and net migration rate, urbanization, and currently married women.
8. ***K-Nearest Neighbors****Best Obtained Model:* ***After hyperparameter tuning -*** *{'n\_neighbors': 3}  
   Mean Squared Error: 0.23698237547892714  
   Mean Absolute Error: 0.2809195402298851  
   R2 Score: 0.8299063396041326*This model also made good sense to implement, as countries that are statistically similar to each other in various measurements tend to have similar population growth rates (for example, Niger and Syria consistently rank among the top in birth rate and total fertility rate and thereby have a similar population growth rate). The KNN model also worked very well after scaling the data (helps remove noise), producing similar evaluation results to the gradient boosting regressor. The optimal hyperparameter was using k=3, so the 3 closest data points (countries) influenced the prediction for a new data point.
9. ***XGBoost****Mean Squared Error: 0.27650118854663175  
   R2 Score: 0.8284332947387901*

I implemented this model after learning about it during my online research. I wanted to see if there could be any new insights gained by trying XGBoost, a regularizing gradient boost framework. The model works about as well as gradient boosting, and was very interesting to read about and play around with.

All the models after some tinkering (scaling, selecting k best features, or hyperparameter tuning) produced good results, but in general, **the linear models could not achieve the goal of minimizing errors as well as the non-linear models**. Some key takeaways:

* The dataset was not too complex, but complex enough that linear models were not the best; they required feature scaling or selecting to reduce complexity
* For the most part, the best features included total fertility rate, birth rate, death rate, net migration rate, and median age.
* Some other features that had an influence were urbanization, currently married women, life expectancy at birth, and others.
* Features that did not appear include total population, total land area, and agricultural land use. This was a bit surprising, but it makes overall sense that these features would take the backseat to more influential factors like birth and death rates.
* Bagging and boosting really improved the efficiency of the decision tree models, feature selection and scaling worked best for the linear models, while scaling did the best for K-Nearest Neighbors

Based on the results, the best models were consistently the tree-based models (decision tree, random forest, gradient boosting). They also fit the goal of finding the best features much better than the linear models. The **gradient boosting regressor** and **k-nearest neighbors** were the best performers, while the **random forest** and **decision tree** were close seconds.

**Conclusion**

Through these results, it can be seen that non-linear models handle datasets with multicollinearity much better than linear models. Although the dataset was reduced in size with feature engineering, the features were noisy and had large standard deviations in general, causing simpler models to perform worse without any scaling or feature selection. Through this project, I was able to get an understanding of the weaknesses of the models (ex. linear models, KNN, not performing well with default hyperparameters) and the many methods to improve the models (feature scaling, feature selection, grid search). The strengths of the non-linear models over the linear models on the dataset were highlighted by the implementation of the project and spoke volumes as to how powerful bagging/boosting and greedy algorithms (for decision trees) are. Overall, this project strengthened my understanding of the strengths and weaknesses of several models, and also inspired me to learn more about the wide world of machine learning.

**Citations**

1. *Pew Research Center*, assets.pewresearch.org/wp-content/uploads/sites/11/2015/03/PF\_15.04.02\_ProjectionsFullReport.pdf. Accessed 16 Nov. 2023.
2. Team, DataCamp. “Lasso and Ridge Regression in Python Tutorial.” *DataCamp*, DataCamp, 25 Mar. 2022, www.datacamp.com/tutorial/tutorial-lasso-ridge-regression.
3. “XGBoost Python Packageℑ.” *XGBoost Python Package - Xgboost 2.0.2 Documentation*, xgboost.readthedocs.io/en/stable/python/index.html. Accessed 7 Dec. 2023.